

The Sourceror's Apprentice

The Assembly Language Journal of Merlin Programmers

Vol. 1 No. 5 May, 1989

News, Views, and Much Ado About 8-bit Text Editing

I'll sure be glad when Ross's Great Cross Country Moving Adventure gets finished. What a pain in the circuitry. Let's see, I'd best remind you that we have a new address:

Ariel Publishing
P.O. Box 398
Pateros, WA 98846

(509) 624-3161

I'll be unavailable from May 28th - June 10th, too. I apologize in advance to those who find that an inconvenience.

Furthermore, I promise that we'll get caught up on back ordered stuff (mostly back issues) ASAP. I'll have an authentic, flesh and blood, full time secretary beginning June 10th, so we should really start to resemble a professional operation soon (knock on wood). Don't fret too much about us losing your orders or correspondence. They're all tucked away in my overstuffed briefcase. Isn't that reassuring?

If it sounds like we're busy here at the Ariel igloo, that's 'cuz we are. Things are really going pretty well. Though our progress is modest by most standards, our overhead is low, too, so things are moving steadily forward.

I've received a jillion suggestions about topics to cover in **The Apprentice**, all of them good. We've got article fodder for the next few years, I think. Feel free to contribute a suggestion or two - I read 'em all. This month's coverage of text editing routines is brought to you courtesy of intense popular demand (and Prof. Robert Moore, who had no idea how timely his submission was!) You saved my skin again, Bob, and provided a truly outstanding set of routines for the readership. I've never seen anything this comprehensive published anywhere. I



Packing up the Ariel igloo

am proud to bring it to y'all in its entirety this month.

A GS Update

This is really old news, but... I've been known to wax preachy regarding the *Apple II GS Toolbox References*. They're pretty close to indispensable for GS work. At present there are two volumes, but Apple recently released the *Toolbox Reference Update*. The *Update*, too, is finding its way onto my "can't do without" list.

First, it corrects outright errors in the *References*. Although there really aren't that many, some of the existing errors can drive you nuts. The *QuickDraw* chapter, for example, says that calls such as `_LineTo` and `_MoveTo` take global coordinates. It probably didn't take many of you GS types too long to figure out that they don't.

The *Update* also lists several new calls added to the toolboxes since the manuals went to press. We've already examined one of these within our pages, `_AlertWindow`. Another useful new routine is called `_RealFreeMem`, and it's worth a quick "once over" here.

As you've probably already discovered, the Memory Manager function `_FreeMem` only returns the amount of memory *not currently in use*. This is sensible, of course, except when we need to know how much

more memory would be available if purgable blocks were evicted from the joint.

Thence cometh `_RealFreeMem`. This new call will dutifully report the amount of memory available *after* purgeable blocks are removed. As the *Update* suggests, it gives a much more accurate picture of the state of the silicon. Note that it does *not* actually execute a purge, it just reports what would things would be like if one happened.

The following snippet shows how to use the call:

```
* _RealFreeMem call

PushLong #0          ;result space
_RealFreeMem
PullLong FreeBytes

* to convert to kilobytes

lda FreeBytes+1
lsr
lsr
sta FreeKilobytes
```

The conversion to kilobytes code looks odd at first blush, but stop and consider that converting from bytes to kilobytes entails a division by 1024. If you're thinking in terms of binary shifts to the right, each of which is a division by two, dividing by 1024 means ten shifts to the right (LSRs). The lowest byte, then, is lost completely. It would be shifted into nothingness.

By leaving the lowest byte out of the process altogether and starting to work on `FreeBytes+1`, we save a few bytes, a few instructions, and a few cycles. This is never a bad idea when possible, even on the memory rich GS.

Note, too, that the high byte of the four byte variable `FreeBytes` is ignored, this because it must always be equal to zero on the GS (at least when we're talking about the range of memory locations).

By the way, if you want to add a macro for this call to your MEM.MACS library on the Merlin disk, make it look like this:

```
~RealFreeMem MAC
    PHS 2
    _RealFreeMem MAC
    Tool $2F02
    <<<
```

(This macro is already in later version of Merlin 8/16 and in the new Merlin 16+. I've been asked to remember those who don't have the "latest and greatest" versions of Merlin. The above macro is in their honor.)

Back at the ranch, I've only scratched the surface. The entire *Update* is packed with goodies that make 16 bit life easier. It is available for \$30 from APDA (800/282-2732). Yes, \$30 is a bit much for looseleaf material. But that is a debate for another day (a day that is coming all too quickly, it appears).

Another product I recommend is RavenWare's *DesignMaster*. Author Chris Haun has put together a neat code generating utility which lets you literally draw your windows, dialogs, menus, etc. Priced at \$30, the package is a genuine d-e-a-l. You draw it, and *DesignMaster* produces the code and definition data in either APW or Merlin format (for assembly language junkies), or C or Forth for you high level types. (RavenWare, 23930 Ocean Avenue, #201, Torrance, CA 90505).

AppleFest attendees were also wowed by another code generating product due out in September. GENESYS supposedly does everything except press keys for you. It had better, with a price tag of \$125. Seriously, though, my 'Fest spies say it looks *very* impressive.

The GS marketplace is warming. That alone is neat, but Apple's literal "pre-announcement" of System Disk 5.0 at AppleFest bodes well for the II, too. The Apple II is *never* going to get the support I think it merits, but I'll devour any bones I'm thrown (and continue yapping for more).

Enough news and views. On with Professor Moore's show... I think you'll like it. And there are no commercial interruptions!

&Input, &Print, and &Get OR More Bang for Your Text Bytes

by Robert C. Moore
1204 Marton Street
Laurel, MD 20707

Editor: These routines put advanced and powerful text editing routines right at your fingertips. It's the best and most comprehensive program of its kind that I've ever seen.

Bob chose to connect his program to Applesoft, but it is possible to take the ampersand and variable passing routines out if you want to operate in a "pure" assembly environment. It would be a tad trickier, though, if you wanted to switch out the Applesoft ROMs altogether.

I hope you enjoy Bob's code as much as I have.

This article documents an Applesoft extension program which I have called INPUT.PRINT.GET. The program adds three ampersand commands to Applesoft:

&INPUT x\$,
&PRINT x\$, and
&GET x\$.

The commands behave much as the similar commands in AppleWorks' SU2.OBJ do.

The source code is in a format that is compatible with most 6502 assemblers, including Merlin; it needs very few modifications to be used with most other popular assemblers. The source code is very heavily commented. This is to facilitate customization by readers of **The Sourcerer's Apprentice** who choose to modify the program for their own special uses.

The comments in the source code carefully document the program's use. They also should help you to understand how various portions of the program work. Specifically, the source code illustrates how to install machine language routines above HIMEM in

both DOS3.3 and ProDOS 8, how to chain into the ampersand hook, how to read the value of an Applesoft real variable from machine language, how to set the value of an Applesoft string or real variable from machine language, and how to use software "switches" and "signatures" to obtain multiple functions using a single module of code.

The three ampersand commands are installed simply by BRUNning INPUT.PRINT.GET prior to assigning any string variables. (Under ProDOS 8 and BZSIC.SYSTEM you may use the smart run [dash] command.) Once installed, the object code uses only 1024 bytes of memory. During installation, locations \$2096 - \$24FF are used temporarily. The source code explains how this temporary workspace may be relocated, if the location I have chosen conflicts with any of your previously installed programs.

Zero-page locations \$3C through \$47 are used temporarily by INPUT.PRINT.GET. Their original contents are destroyed. (This should not be a problem, because these are scratchpad locations for ProDOS 8 and the system monitor.) All other zero-page locations that are normally available to assembly language programs remain accessible.

I have attempted to make this program easy to use and as compatible as possible with other enhancements to Applesoft. The program has been tested on an Apple //c, a "regular" IIe, an enhanced IIe, and a IIGS. It assumes you have Applesoft in ROM, and that you are using text page 1 in either 40- or 80-column mode.

&INPUT x\$

&INPUT x\$ prints the current (default) value of the specified string variable x\$ to the current text screen window (40- or 80-column display) and then permits you to edit the string from the keyboard.

The powerful string editing features of the "&INPUT x\$" command are particularly useful:

ARROW KEYS move the blinking underscore "insert" cursor. If the edit string occupies more than one line in the text window then the up- and down-arrow keys will work. This gives you full-screen editing of the string.

DELETE deletes the character to the left of the cursor and closes up the resulting gap in the edit string.

CTRL-D deletes the character under the cursor and closes up the resulting gap in the edit string.

CTRL-X ("cross out") erases the entire edit string.

CTRL-Y erases from the cursor to the end of the edit string.

CTRL-B moves the cursor to the beginning of the edit string.

CTRL-N moves the cursor to the end of the edit string.

CTRL-C toggles the case of the character under the cursor, if it is a letter (alphabetic character), then advances the cursor to the right. Upper case letters are converted to lower case; lower case letters are converted to upper case.

RETURN accepts the current edit string, strips off any trailing spaces, and assigns the resulting string as the new value for the specified string variable, x\$.

ESCAPE aborts the &INPUT x\$. The value of the specified string variable, x\$, remains at the default. The Applesoft real variable ES is set to 1. (If ESCAPE is not used to abort an &INPUT x\$, the value of variable ES will be set to 0.) The abort may be detected following &INPUT x\$ by using ON ES GOTO.

OPEN-APPLE (when used to modify another key) aborts &INPUT x\$ and sets the Applesoft variable OA to 128 plus the ASCII value for the key that was pressed (i.e., high-ASCII). (If OPEN-APPLE-key is not used to abort &INPUT x\$, the value of variable OA

will be set to zero.) For example, OPEN-APPLE-A will abort &INPUT x\$ (the value of x\$ will remain at the default) and set the value of variable OA to 193. Use of the OPEN-APPLE key to abort &INPUT x\$ may be detected by using IF OA GOTO.

SOLID-APPLE (when used to modify another key) aborts &INPUT x\$ and sets the Applesoft variable SA to 128 plus the ASCII value for the key that was pressed (i.e., high-ASCII). (If SOLID-APPLE-key is not used to abort &INPUT x\$, the value of variable SA will be set to zero.) If both the OPEN-APPLE and the SOLID-APPLE keys are used to modify another key, then both OA and SA will be assigned the high-ASCII value of the key that was pressed.

Another Applesoft variable, FL, may be used to set the maximum field length; that is, the value of FL will determine the maximum length for the edit string. For example, if you are using &INPUT x\$ to input a filename under ProDOS, you would want to set FL = 15 because that is the maximum length of a ProDOS filename. If, during editing, you attempt to increase the length of the edit string beyond the value of FL, you will be beeped. If you execute &INPUT x\$ with a default value for x\$ that is greater in length than the value of FL, you will generate an Applesoft STRING TOO LONG error. You will get the same error (STRING TOO LONG) if your default string is so long that the top line scrolls off the top of the text screen window as the string is printed. If FL = 0, the maximum field length will be 255 characters.

&GET x\$

&GET x\$ works as the &INPUT x\$ command does, except that the string is limited to exactly one character, no default string is displayed on screen, and ESCAPE may not be used to abort. The Applesoft variables OA and SA work as with &INPUT x\$. Following &GET x\$, the high-ASCII value of the key that was pressed may be retrieved from address \$3C = 60 using PEEK(60). The new value of x\$ will be the single character that was typed at the keyboard.

&GET x\$ may be used to get any encoded keypress except CTRL-RESET or OPEN-APPLE-CTRL-RESET. To determine if

ESCape was pressed during &GET x\$, use ON (PEEK(60) = 155) GOTO.

As with &INPUT x\$, use of the OPEN-APPLE or SOLID-APPLE keys may be detected using IF OA GOTO and/or IF SA GOTO.

While &INPUT x\$ and &GET x\$ are waiting for keystrokes, they advance a 16-bit unsigned integer in locations \$4E,\$4F (78, 79) to a new "random" value. (This value may be used to "seed" a pseudorandom number generator.) The "random" value may be obtained using PEEK(78) + 256 * PEEK(79).

&PRINT x\$

&PRINT x\$ prints the current value of the specified string variable, x\$, to the text window with word-wrapping. Lines are broken at spaces, if possible. &PRINT x\$ leaves the text screen cursor immediately to the right of the last character that was printed.

I believe this program will be of great interest to readers of The Sourceror's Apprentice, most of whom are intermediate-level Apple II programmers who delight in finding new ways by which the power of Apple II assembly language may be released in their own programs.

```

1 ***** This routine adds three ampersand
2 ***** commands to Applesoft. The first,
3 ** &INPUT x$, is a "defaulted input
4 ** DEFAULTED ** almost anything" command that
5 ** INPUT ** inputs up to 255 characters to any
6 ** ** string variable x$. The maximum
7 ** WORD-WRAP ** number of characters in the edit
8 ** PRINT ** string is set by the value of the
9 ** ** variable FL. The current value of
10 ** GOOD GET$ ** x$ is the default. The default
11 ** ** string may be edited, then accepted
12 ***** by pressing <RETURN>. The INPUT may
13 ***** be aborted by pressing <ESC>, which
14 * will set the value of variable ES to
15 *PUBLIC DOMAIN one. The &INPUT also may be aborted
16 *APPLE // UTILITY by pressing one of the apple keys in
17 * written for conjunction with another key, in
18 * "Reboot" and which case variable OA or SA will be
19 * The Sourceror's assigned the value of the key that
20 * Apprentice was pressed. The second command,
21 * by &GET x$, inputs a single keystroke.
22 *Robert C. Moore Control codes may be entered using
23 *1204 Marton St. &GET x$, and OA and SA work as
24 *Laurel, MD 20707 with &INPUT x$. The third command,
25 * &PRINT x$, prints x$ with word-wrap.
26 *Most recent code Both 40- and 80-column text screens
27 *update was done: are supported, and the boundaries
28 *March 29, 1989 of the text window are observed.
29 ;
30 *Assembled using 6502 opcodes only
31 ;
32 *Compatible with all Apple II computers
33 ;
34 *Compatible with ProDOS 8
35 ;
36 *Compatible with DOS 3.3
37 ;
38 *Zero-page usage
39 ;
=000D 40 CHARAC EQU $0D ;String term for STRLT2
=000E 41 ENDCHR EQU $0E ;String term for STRLT2
=0010 42 DIMFLG EQU $10 ;Dimension flag in PTRGET
=0011 43 VALTYP EQU $11 ;Numeric: 0; String: $FF

```

```

=0012 44 INTFLG EQU $12 ;$80 if integer, else $00
=0020 45 WNDLFT EQU $20 ;Text window left
=0021 46 WNDWID EQU $21 ;Text window width
=0022 47 WNDTOP EQU $22 ;Text window top
=0023 48 WNDBOT EQU $23 ;Text window bottom + 1
=0024 49 CH EQU $24 ;40-col horizontal cursor
=0025 50 CV EQU $25 ;40-col vertical cursor
=0028 51 TBASE EQU $28 ;Text base address
=003C 52 SOURCE EQU $3C ;Source address for move
53 ;
=003C 54 KEYCOD EQU $3C ;OA, SA or GET keycode
=003D 55 BOTCV EQU $3D ;Bottom display CV
=003E 56 BOTCH EQU $3E ;Bottom display CH
=003F 57 OLDCV EQU $3F ;Old vertical cursor
=0040 58 OLDCH EQU $40 ;Old horizontal cursor
=0041 59 FLDLEN EQU $41 ;Maximum field length
=0042 60 DEST EQU $42 ;Dest. address for move
=0042 61 STRLEN EQU $42 ;String length
=0043 62 TOPCV EQU $43 ;V cursor for top
=0043 63 OAFLAG EQU $43 ;Open-apple flag
=0044 64 TOPCH EQU $44 ;H cursor for top
=0044 65 SAFLAG EQU $44 ;Solid-apple flag
=0045 66 SWITCH EQU $45 ;Software switch
=0046 67 ESCFLG EQU $46 ;Escape flag
=0046 68 TEMPX EQU $46 ;X-reg temporary store
=0047 69 TEMPY EQU $47 ;Y-reg temporary store
=004E 70 RANDOM EQU $4E ;Random number
=006F 71 FRETOP EQU $6F ;Bottom of string storage
=0073 72 HIMEM EQU $73 ;Top of free memory
=0081 73 VARNAM EQU $81 ;Variable name
=0083 74 VARPNT EQU $83 ;Variable pointer
=0085 75 FORPNT EQU $85 ;Destination string addr
=00AB 76 STRNG1 EQU $AB ;String pointer #1
77 ;
78 ;
79 ; Buffer for edit string
80 ;
=0200 81 EDBUF EQU $200 ;Buffer for edit string
82 ;
83 *Notice that because this program uses the input
84 *buffer as a workspace in which to form the edit
85 *string, calls to this program from immediate mode
86 *will almost always end in a ?SYNTAX ERROR. This
87 *program was designed for use in deferred mode only.
88 ;
89 ;
90 ; Ampersand hook
91 ;
=03F5 92 AMPERH EQU $3F5 ;Ampersand hook
93 ;
94 ;
95 ; Screen hole usage
96 ;
=057B 97 CH80 EQU $57B ;80-col horizontal cursor
98 ;
99 ;
100 ; BASIC.SYSTEM entry points
101 ;
=BE09 102 ERROUT EQU $BE09 ;BASIC error handler
=BEF5 103 GETBUFR EQU $BEF5 ;Get buffer space
104 ;
105 ;
106 ; ProDOS entry point
107 ;
=BF00 108 PROMLI EQU $BF00 ;ProDOS M.L. Interface
109 ;
110 ;

```

```

111                                     ; Hardware page usage
112                                     ;
=C000 113 KEYBD EQU $C000             ;Keyboard data & strobe
=C001 114 STORE80 EQU $C001          ;PAGE2 switches 1 and 1X
=C010 115 STROBE EQU $C010           ;Clear keyboard strobe
=C01F 116 RD80COL EQU $C01F          ;Read 80-col switch
=C054 117 PAGE1 EQU $C054            ;Select page 1
=C055 118 PAGE2 EQU $C055            ;Select page 2 (or 1X)
=C061 119 READOA EQU $C061           ;Read 'open-apple key
=C062 120 READSA EQU $C062           ;Read solid-apple key
121                                     ;
122                                     ;
123                                     ; Applesoft entry points
124                                     ;
=00B1 125 CHRGET EQU $00B1           ;Get next character
=00B7 126 CHRGET EQU $00B7           ;Get current character
=D412 127 ERROR EQU $D412            ;Process error code in X
=D539 128 GDBUFS EQU $D539           ;Form string in EDBUF
=EB27 129 STORE EQU $EB27            ; (FAC) to real variable
130                                     ;at address FORPNT
=DA7B 131 PERMST EQU $DA7B           ;Make temp str permanent
=DD6C 132 CHKSTR EQU $DD6C           ;Check for string var
=DEC9 133 SYNERR EQU $DEC9           ;Report syntax error
=DFE3 134 PTRGET EQU $DFE3           ;Get pointer to variable
=E04F 135 VARLOC EQU $E04F           ;Locate real variable
=E301 136 SNGFLT EQU $E301           ;Float unsigned int (Y)
=E3ED 137 STRLT2 EQU $E3ED           ;Build string descriptor
=E6FB 138 CONINT EQU $E6FB           ;Convert (FAC) to byte
=EAF9 139 MOVFM EQU $EAF9            ;Move (Y,A) into FAC
140                                     ;
141                                     ;
142                                     ; Applesoft keyword tokens
143                                     ;
=0084 144 INPTKN EQU $84             ;Token for "INPUT"
=00BA 145 PRNTKN EQU $BA             ;Token for "PRINT" or "?"
=00BE 146 GETTKN EQU $BE            ;Token for "GET"
147                                     ;
148                                     ;
149                                     ; Monitor entry points
150                                     ;
=FBDD 151 BEEP EQU $FBDD             ;Beep speaker
=FC22 152 VTAB EQU $FC22             ;Vertical tab
=FD0D 153 COUT EQU $FD0D             ;Output a character
154                                     ;
155                                     ;
156                                     ; Initial load address for main program
157                                     ;
=2100 158 INITAD EQU $2100           ;Initial load address
159                                     ;for main program must
160                                     ;be on a page boundary
161                                     ;(i.e., $xx00).
162                                     ;
163                                     ;
164                                     ;Length of installation code
165                                     ;
=006A 166 INSTAL EQU $6A             ;Installer length
167                                     ;
168                                     ;
169                                     ORG INITAD-INSTAL ;Initial load address
170                                     ; for object code
171                                     ;
172                                     ;
173 *During installation the installation code and the
174 *main program are BLOAded into INITAD-INSTAL. The
175 *memory from that location through INITAD+$3FF is
176 *used temporarily. The value of INITAD should be
177 *chosen so that the installation process doesn't
178 *clobber anything important. As an example, if

```

```

179 *INITAD=$2100, memory from $2096 through $24FF will
180 *be used as a temporary buffer during installation.
181 ;
182 ;
183 ; INSTALLATION CODE
184 ;
185 *The installer lowers HIMEM by $400 (DOS3.3) or re-
186 *quests a 4-page buffer (ProDOS BASIC.SYSTEM). The
187 *main program then is relocated above HIMEM, and the
188 *Applesoft ampersand hook is vectored to it. (The
189 *hook is chained to whatever ampersand routines
190 *were installed previously.) The main program re-
191 *duces the amount of free memory by 1024 bytes.
192 *Under ProDOS, a call to FREEBUFR ($BEF8) will re-
193 *move this program from memory without resetting
194 *the ampersand hook at $3F5; so if you "disinstall"
195 *by calling FREEBUFR (CALL 48888), be very careful
196 *to reset the ampersand hook! No peace-loving
197 *human being ever calls FREEBUFR, unless it is to
198 *disinstall a block of code he himself recently
199 *installed. A word to the wise is sufficient.
200 ;
201 *To install the program, simply execute the following
202 *(this assumes the object file is INPUT.PRINT.GET):
203 *DOS3.3 command: PRINT CHR$(4);"BRUN INPUT.PRINT.GET"
204 *or with ProDOS: PRINT CHR$(4);"-INPUT.PRINT.GET".
205 ;
206 *Notice that, under DOS3.3, the pointer to the bottom
207 *(of string storage (FRETOP) will be set equal to the
208 *pointer to the top of string storage (HIMEM). This
209 *assumes that no strings have been created at the
210 *time the installation code is executed. Make sure
211 *that the BRUN INPUT.PRINT.GET command is executed
212 *before any strings have been created.
213 ;
002096: AD 00 BF 214 LDA PROMLI ;Are we under ProDOS?
002099: C9 4C 215 CMP #$4C ;JMP op-code if ProDOS
00209B: F0 11 =20AE 216 BEQ PRODOS
217 ;
00209D: 38 218 SEC ;It's DOS3.3, so
00209E: A5 74 219 LDA HIMEM+1 ;lower HIMEM by $400.
0020A0: E9 04 220 SBC #4
0020A2: 85 74 221 STA HIMEM+1
0020A4: 85 70 222 STA FRETOP+1 ;FRETOP too!
223 ;
224 *(Assumes no string assignments have been made.)
225 *Accumulator now holds high byte of buffer addr.
226 ;
0020A6: A0 00 227 LDY #0 ;Force low byte to zero
0020A8: 84 73 228 STY HIMEM ;to simplify relocation.
0020AA: 84 6F 229 STY FRETOP
0020AC: F0 0A =20B8 230 BEQ L0 ;Always taken
231 ;
0020AE: A9 04 232 PRODOS LDA #4 ;Request 4 256-byte pages
0020B0: 20 F5 BE 233 JSR GETBUFR ;using GETBUFR.
234 ;
235 *Accumulator now holds high byte of buffer addr.
236 ;
0020B3: 90 03 =20B8 237 BCC L0 ;Continue if no error,
0020B5: 4C 09 BE 238 JMP ERROUT ;else exit thru ERROUT.
239 ;
0020B8: AC F5 03 240 L0 LDY AMPERH ;Chain into the
0020BB: 8C 14 21 241 STY OLDHOOK ;ampersand hook.
0020BE: AC F6 03 242 LDY AMPERH+1
0020C1: 8C 15 21 243 STY OLDHOOK+1
0020C4: AC F7 03 244 LDY AMPERH+2
0020C7: 8C 16 21 245 STY OLDHOOK+2
0020CA: 8D C1 23 246 STA JMP1+2 ;Fix JMP instructions

```



```

0020CD: 8D C6 23      247      STA    JMP2+2
248                      ;
249 * JMP1 and JMP2 are the only
250 * instructions in the main program
251 *that reference addresses in the
252 *first page of the main program.
253 *The high-order bytes of these
254 *addresses need to be adjusted.
255                      ;
0020D0: 8D F7 03      256      STA    AMPERH+2 ;Fix ampersand hook.
0020D3: 18             257      CLC
0020D4: 69 01          258      ADC    #1      ;Step to 2nd page of main
0020D6: 8D A2 23      259      STA    JMP3+2 ;Fix JMP instructions
260                      ;
261 *JMP3 is the only instruction in
262 *the main program that references
263 *an address in the second page of
264 *the main program. The high-order
265 *byte of this address needs to be
266 *adjusted.
267                      ;
0020D9: 69 02          268      ADC    #2      ;Step to 3rd page of main
0020DB: 85 43          269      STA    DEST+1 ;DEST=BUFFER+$300
0020DD: A9 24          270      LDA    #>INITAD+$300 ;SOURCE=INITAD+$300
0020DF: 85 3D          271      STA    SOURCE+1
0020E1: A0 00          272      LDY    #0      ;Initialize index
0020E3: 84 42          273      STY    DEST
0020E5: 84 3C          274      STY    SOURCE
0020E7: 8C F6 03      275      STY    AMPERH+1
0020EA: A9 4C          276      LDA    #$4C    ;JMP op-code
0020EC: 8D F5 03      277      STA    AMPERH ;(Just to be certain!)
0020EF: A2 04          278      LDX    #4      ;Move 4 256-byte pages
0020F1: B1 3C          279      MOVE   LDA    (SOURCE),Y ;Get a byte
0020F3: 91 42          280      STA    (DEST),Y ;Relocate it
0020F5: C8             281      INY
0020F6: D0 F9 =20F1    282      BNE    MOVE ;Back until page is done
0020F8: C6 3D          283      DEC    SOURCE+1 ;Step to next page
0020FA: C6 43          284      DEC    DEST+1
0020FC: CA             285      DEX ;Decrement page counter
0020FD: D0 F2 =20F1    286      BNE    MOVE ;Back if not done
0020FF: 60             287      RTS ;Installation complete!
288                      ;
289                      ;
290                      ;      MAIN PROGRAM
291                      ;
292 *The main program parses the text that follows the
293 *ampersand and responds accordingly. If an INPUT,
294 *GET, or a PRINT token is not found, control is
295 *passed to any previously installed ampersand routine.
296 *With &GET x$, the current value of x$ is not printed.
297 *With &INPUT x$ and &PRINT x$, the current value of
298 *x$ is printed to the text screen window beginning
299 *at the current cursor location, with (&PRINT) or
300 *without (&INPUT) word-wrapping. With &INPUT, this
301 *default string then may be edited using the blinking
302 *underscore "insert" cursor and the following keys:
303                      ;
304 *ARROW keys move the blinking underscore cursor.
305 *If the string occupies more than one line,
306 *the up- and down-arrow keys will work.
307 *DELETE deletes character to left of cursor.
308 *CTRL-D deletes character under the cursor.
309 *CTRL-X erases the edit string.
310 *CTRL-Y clears from cursor to end of edit string.
311 *CTRL-B moves cursor to beginning of edit string.
312 *CTRL-N moves cursor to end of edit string.
313 *CTRL-C toggles the case of the character under

```

```

314 *the cursor, if it is a letter:
315 *upper case letters are converted to lower;
316 *lower case letters are converted to upper.
317 *RETURN accepts the current edit string and
318 *assigns it to the variable, x$.
319 *ESC aborts with a variable (ES) set to "1".
320 *OPEN-APPLE (in conjunction with another key)
321 *aborts with a variable (OA) set to
322 *the code for the key that was pressed.
323 *SOLID-APPLE (in conjunction with another key)
324 *aborts with a variable (SA) set to
325 *the code for the key that was pressed.
326 ;
327 *If the specified string, x$, has a length that ex-
328 *ceeds the specified maximum field_length, FL, then a
329 *STRING TOO LONG error will be generated. The same
330 *error will be generated if the edit string ever
331 *grows so long that its top line scrolls out of the
332 *text window. The text window must be at least two
333 *characters wide. If &INPUT is aborted by pressing
334 *<ESC>, this may be detected using an ON ESCAPE GOTO
335 *statement. An apple-key combination may be detected
336 *using an IF OA GOTO or IF SA GOTO statement. When
337 *nonzero, the value of OA or SA is the hi-ASCII
338 *keycode. All three ampersand routines leave the
339 *text screen cursor just beyond the end of the
340 *printed string. A blinking underscore cursor is
341 *used during &GET and &INPUT editing. When control
342 *returns to Applesoft, the text cursor always will
343 *be restored to whatever cursor was in use at the
344 *time the ampersand routine was invoked. If no
345 *FL variable was defined prior to &INPUT, or if
346 *the value of FL had been set equal to zero, the
347 *field length defaults to 255 characters. Zero-page
348 *locations $3C through $47 are used temporarily by
349 *this program; their original contents are destroyed.
350 *If the variables FL, ES, OA, and SA do not exist
351 *prior to invoking &INPUT, &GET, or &PRINT, they
352 *will be created for you, and FL will default to
353 *zero (which indicates field_length = 255). When
354 *terminated by <RETURN>, &INPUT strips trailing
355 *spaces from the edit string before making a new x$.
356 *Following &GET, PEEK(60) will yield the Hi-ASCII
357 *code for the character in x$; PEEK(60)-128 will
358 *give the Lo-ASCII code. &GET always clears the
359 *variable ES to zero, even if the key that was
360 *"gotten" was <ESC>. OA and SA behave exactly the
361 *same with &GET as they do with &INPUT, except that
362 *the GET is not aborted. If you &GET an apple key
363 *combination, x$ receives the character, location
364 *60 receives the Hi-ASCII code, and OA and/or SA
365 *also receive(s) the Hi-ASCII code. To determine
366 *if <ESC> was pressed during an &GET, use
367 *ON (PEEK(60)=155) GOTO instead of ON ESCAPE GOTO.
368 *&INPUT will respond to all ASCII codes except
369 *control codes and DELETE.
370 ;
002100: 20 B7 00 371 JSR CHRGOT ;Get character after "&"
002103: A2 FF 372 LDX #$FF ;Flag value for &INPUT
002105: C9 84 373 CMP #INPTKN ;Compare to INPUT token
002107: F0 0E =2117 374 BEQ L1 ;If &INPUT, SWITCH=$FF
002109: E8 375 INX ;Flag for &PRINT ($00)
00210A: C9 BA 376 CMP #PRNTKN ;Compare to PRINT token
00210C: F0 09 =2117 377 BEQ L1 ;If &PRINT, SWITCH=$00
00210E: A2 FE 378 LDX #$FE ;Flag for &GET ($FE)

```

```

002110: C9 BE      379      CMP    #GETTKN    ;Compare to GET token
002112: F0 03 =2117  380      BEQ    L1        ;If &GET, SWITCH=$FE
002114: 4C C9 DE      381  OLDHOOK JMP    SYNERR    ;Old &-hook stored here
                                382      ;
002117: 86 45      383  L1      STX    SWITCH    ;Set switch
                                384      ;
                                385  *SWITCH = $00 for &PRINT
                                386  *SWITCH = $FE for &GET
                                387  *SWITCH = $FF for &INPUT
                                388      ;
002119: A0 00      389      LDY    #0        ;Default STRLEN to 0
00211B: 84 42      390      STY    STRLEN
                                391      ;
00211D: 84 10      392      STY    DIMFLG    ;Initialize flags
00211F: 84 11      393      STY    VALTYP    ;Numeric, not string
002121: 84 12      394      STY    INTFLG    ;Real, not integer
                                395      ;
                                396  ***Find value of variable FL***
002123: A9 46      397      LDA    #$46      ;Lo-ASCII 'F'
002125: 85 81      398      STA    VARNAM
002127: A9 4C      399      LDA    #$4C      ;Lo-ASCII 'L'
002129: 85 82      400      STA    VARNAM+1
00212B: 20 4F E0    401      JSR    VARLOC    ;Locate the variable FL
00212E: 20 F9 EA    402      JSR    MOVFM    ;Move (Y,A) to FAC
002131: 20 FB E6    403      JSR    CONINT    ;Integer in X reg
002134: 8A          404      TXA          ;Examine value of FL
002135: D0 01 =2138  405      BNE    STORFL
002137: CA          406      DEX          ;Default to 255
002138: 86 41      407  STORFL STX    FLDLEN    ;Store field_length
                                408      ;
                                409  ***Locate string variable***
00213A: 20 B1 00    410      JSR    CHRGET    ;Advance TXTPTR
00213D: 20 E3 DF    411      JSR    PTRGET    ;Get ptr to str descript
002140: 85 85      412      STA    FORPNT    ;Save pointer in FORPNT
002142: 84 86      413      STY    FORPNT+1 ;for later use by PERMST.
002144: 20 6C DD    414      JSR    CHKSTR    ;Check for string var
002147: 20 B7 00    415      JSR    CHRGOT    ;Examine next character
00214A: F0 03 =214F  416      BEQ    SYNTOK    ;Branch if : or EOL
00214C: 4C C9 DE    417      JMP    SYNERR    ;Error if not : or EOL
                                418      ;
00214F: A5 24      419  SYNTOK LDA    CH        ;Update text cursor
002151: F0 03 =2156  420      BEQ    L2        ;Update CH80 only if
002153: 8D 7B 05    421      STA    CH80      ;CH > 0.
002156: A4 25      422  L2      LDY    CV        ;
002158: 2C 1F C0    423      BIT    RD80COL    ;80-column display?
00215B: 10 03 =2160  424      BPL    SAVCUR
00215D: AD 7B 05    425      LDA    CH80
002160: 84 43      426  SAVCUR STY    TOPCV    ;Cursor values for
002162: 85 44      427      STA    TOPCH    ;start of string
                                428      ;
002164: A0 00      429  REDO    LDY    #0        ;Initialize index
002166: 84 47      430      STY    TEMPY
002168: A6 45      431      LDX    SWITCH
00216A: E0 FE      432      CPX    #$FE      ;Is this &GET?
00216C: F0 04 =2172  433      BEQ    GETPNT    ;If so, use default.
00216E: B1 83      434      LDA    (VARPNT),Y ;Get length of string
002170: 85 42      435      STA    STRLEN    ;Store in string_length
002172: C8          436  GETPNT INY          ;Step to next character
002173: B1 83      437      LDA    (VARPNT),Y ;LOB of pointer
002175: 85 AB      438      STA    STRNG1    ;STRNG1 points to x$
002177: C8          439      INY          ;Step to next character
002178: B1 83      440      LDA    (VARPNT),Y ;HOB of pointer
00217A: 85 AC      441      STA    STRNG1+1

```

00217C: A0 00	442		LDY	#0	;Reset index
00217E: A5 45	443		LDA	SWITCH	;Test switch
002180: D0 4C =21CE	444		BNE	PRNWRD	;INPUT or GET: go print
002182: 84 47	445	L3	STY	TEMPY	;Else word-wrap
002184: A2 01	446		LDX	#1	;Initialize char count
002186: C0 00	447		CPY	#0	;No leading space at
002188: D0 01 =218B	448		BNE	L4	;start of string.
00218A: CA	449		DEX		
00218B: C4 42	450	L4	CPY	STRLEN	;Reached end of string?
00218D: B0 09 =2198	451		BCS	CHKWRD	;Branch if yes
00218F: C8	452		INX		
002190: E8	453		LDX	(STRNG1),Y	;Increment word length
002191: B1 AB	454		CMP	#\$20	;Lo-ASCII space?
002193: C9 20	455		BNE	L4	;No. Keep going.
002195: D0 F4 =218B	456		SEC		;Yes, prepare to SBC.
002197: 38	457		LDA	WNDWID	;Get window width
002198: A5 21	458	CHKWRD	BIT	RD80COL	;80-column display?
00219A: 2C 1F C0	459		BPL	L5	
00219D: 10 04 =21A3	460		SBC	CH80	;Compute dist to R edge
00219F: ED 7B 05	461		DFB	\$2C	;Skip next instruction
0021A2: 2C	462		SBC	CH	
0021A3: E5 24	463	L5	STX	TEMPX	
0021A5: 86 46	464		TAX		;Save distance to go
0021A7: AA	465		CMP	TEMPX	;Will it fit?
0021A8: C5 46	466		BCS	L7	;Yes. Go print it.
0021AA: B0 08 =21B4	467				;
	468				
0021AC: A9 A0	469	L6	LDA	#\$A0	; (Hi-ASCII space)
0021AE: 20 ED FD	470		JSR	COUT	;Pad with spaces
0021B1: CA	471		DEX		;to end of line.
0021B2: D0 F8 =21AC	472		BNE	L6	
	473				;
0021B4: A4 47	474	L7	LDY	TEMPY	;Restore index
0021B6: F0 16 =21CE	475		BEQ	PRNWRD	;No space if 1st char
0021B8: A5 24	476		LDA	CH	
0021BA: 2C 1F C0	477		BIT	RD80COL	;80-column display?
0021BD: 10 03 =21C2	478		BPL	L8	
0021BF: AD 7B 05	479		LDA	CH80	;Get horiz cursor.
0021C2: C9 00	480	L8	CMP	#0	;At L edge?
0021C4: F0 08 =21CE	481		BEQ	PRNWRD	;Yes. Don't print space.
0021C6: A9 A0	482		LDA	#\$A0	;Hi-ASCII space
0021C8: 99 FF 01	483		STA	EDBUF-1,Y	;Put copy in EDBUF
0021CB: 20 ED FD	484		JSR	COUT	;Print a space
	485				;
0021CE: C4 42	486	PRNWRD	CPY	STRLEN	;At end of string?
0021D0: B0 3D =220F	487		BCS	ENDPRT	;Yes. Finished.
0021D2: B1 AB	488		LDA	(STRNG1),Y	;Get next character
0021D4: 09 80	489		ORA	#\$80	;Convert to hi-ASCII
0021D6: C8	490		INX		;Increment string index
0021D7: A6 45	491		LDX	SWITCH	;Test switch
0021D9: D0 04 =21DF	492		BNE	L9	;No wrap if &INPUT
0021DB: C9 A0	493		CMP	#\$A0	;Hi-ASCII space
0021DD: F0 A3 =2182	494		BEQ	L3	;Go see if next word fits
0021DF: 99 FF 01	495	L9	STA	EDBUF-1,Y	;Put copy in EDBUF
0021E2: A6 25	496		LDX	CV	
0021E4: 20 ED FD	497		JSR	COUT	;Print the character
0021E7: E8	498		INX		
0021E8: E4 23	499		CPX	WNDBOT	;Were we on bottom line?
0021EA: D0 E2 =21CE	500		BNE	PRNWRD	;No, no scroll was done.
	501				;
0021EC: A6 24	502		LDX	CH	;Get horizontal cursor
0021EE: 2C 1F C0	503		BIT	RD80COL	;80-column display?
0021F1: 10 03 =21F6	504		BPL	L10	

```

0021F3: AE 7B 05      505      LDX  CH80      ;Use CH80 instead
0021F6: E0 00      506      CPX  #0        ;Are we at L edge?
0021F8: D0 D4 =21CE  507      BNE  PRNWRD     ;No, no scroll was done.
0021FA: A6 45      508      LDX  SWITCH    ;If &PRINT,
0021FC: F0 D0 =21CE  509      BEQ  PRNWRD     ;keep printing.
510                      ;
511 * We don't care if top line scrolls out of
512 * the text window during &PRINT, but we
513 * must flag it as an error during &INPUT.
514                      ;
0021FE: C6 3F      515      DEC  OLDCV     ;If scroll due to INSERT
002200: C6 43      516      DEC  TOPCV     ;Modify start cursor
002202: 30 06 =220A  517      BMI  L11      ;Error if negative
002204: A6 43      518      LDX  TOPCV     ;
002206: E4 22      519      CPX  WNDTOP    ;Did top scroll off?
002208: B0 C4 =21CE  520      BCS  PRNWRD     ;No. Keep going.
521                      ;
00220A: A2 B0      522      L11      LDX  #$B0     ;Code for STRING TOO LONG
00220C: 4C 12 D4    523      JMP  ERROR     ;Exit thru error process
524                      ;
00220F: A5 24      525      ENDPRT   LDA  CH        ;Store CH in BOTCH
002211: 2C 1F C0    526      BIT  RD80COL    ;80-column display?
002214: 10 03 =2219  527      BPL  STRCH     ;
002216: AD 7B 05    528      LDA  CH80     ;
002219: 85 3E      529      STRCH    STA  BOTCH     ;
00221B: A5 25      530      LDA  CV        ;Store CV in BOTCV
00221D: 85 3D      531      STA  BOTCV     ;
532                      ;
00221F: 24 45      533      TESTSW   BIT  SWITCH    ;Test switch
534                      ;
535 *SWITCH = $00 indicates &PRINT.
536 *SWITCH = $FF indicates initial &INPUT entry.
537 *SWITCH = $FE indicates &GET.
538 *SWITCH = $40 indicates return from <CTRL-R>.
539 *SWITCH = $80 indicates return from <CTRL-X>,
540 *<CTRL-Y>, <CTRL-D>, or INSERT.
541                      ;
002221: 30 03 =2226  542      BMI  L12      ;
002223: 70 09 =222E  543      BVS  CTRLR     ;Return from <CTRL-R>
002225: 60          544      RTS        ;&PRINT is done!
545                      ;
002226: 50 14 =223C  546      L12      BVC  RESTORE    ;Return from <CTRL-Y>,
547                      ;<CTRL-D>, or INSERT.
548                      ;
002228: A5 41      549      LDA  FLDLEN    ;FLDLEN < STRLEN?
00222A: C5 42      550      CMP  STRLEN    ;
00222C: 90 DC =220A  551      BCC  L11      ;Yes. STRING TOO LONG.
552                      ;
00222E: A5 44      553      CTRLR   LDA  TOPCH     ;Cursor to top
002230: 85 40      554      STA  OLDCH     ;
002232: A5 43      555      LDA  TOPCV     ;
002234: 85 3F      556      STA  OLDCV     ;
002236: A0 00      557      LDY  #0        ;Index to beginning
002238: 84 47      558      STY  TEMPY     ;
00223A: 84 46      559      STY  ESCFLG    ;Clear escape flag to "0"
560                      ;
00223C: A0 00      561      RESTORE  LDY  #<EDBUF    ;Aim STRNG1 at EDBUF
00223E: 84 AB      562      STY  STRNG1    ;
002240: A9 02      563      LDA  #>EDBUF    ;
002242: 85 AC      564      STA  STRNG1+1  ;
565                      ;
002244: A5 3F      566      LDA  OLDCV     ;Get previous CV
002246: 85 25      567      STA  CV        ;Store in current CV

```

```

002248: A5 40      568      LDA    OLDCH    ;Get previous CH
00224A: 85 24      569      STA    CH      ;Store in current CH
00224C: 8D 7B 05    570      STA    CH80
                                571      ;
00224F: 20 22 FC    572  GETCHR JSR    VTAB    ;Update TBASE
002252: A4 24      573      LDY    CH      ;Get CH
002254: 8C 7B 05    574      STY    CH80    ;Update CH80
002257: 2C 1F C0    575      BIT    RD80COL  ;80-column display?
00225A: 10 10 =226C 576      BPL    GETCH2
00225C: 8D 01 C0    577      STA    STORE80 ;PAGE2 switches 1 and 1X
00225F: 98          578      TYA
002260: 45 20      579      EOR    WNDLFT  ;LSB=1 if char in main
002262: 4A          580      LSR
                                ;Carry clear if aux
002263: B0 04 =2269 581      BCS    GETCH1
002265: 8D 55 C0    582      STA    PAGE2  ;Select AUX memory
002268: C8          583      INY
                                ;If WNDLFT odd
002269: 98          584  GETCH1 TYA
00226A: 4A          585      LSR
                                ;Compute index
00226B: A8          586      TAY
00226C: B1 28      587  GETCH2 LDA    (TBASE),Y ;Get the character
00226E: 48          588      PHA
                                ;Save original character
00226F: 49 DF      589      EOR    #$DF    ;(Hi-ASCII underscore)
002271: D0 02 =2275 590      BNE    NOZMSK  ;If screen char is "_",
002273: A9 7F      591      LDA    #$7F    ;treat as if space.
002275: 48          592  NOZMSK PHA
                                ;Mask onto stack
002276: 68          593  GETCH3 PLA
                                ;Retrieve mask
002277: 48          594      PHA
                                ;Toggle between
002278: 51 28      595      EOR    (TBASE),Y ;original character
00227A: 91 28      596      STA    (TBASE),Y ;and underscore.
00227C: 2C 00 C0    597  GETCH4 BIT    KEYBD  ;See if key pressed
00227F: 30 12 =2293 598      BMI    GOTKEY
002281: E6 4E      599      INC    RANDOM  ;Use random # as a
002283: D0 F7 =227C 600      BNE    GETCH4  ;flashing cursor timer.
002285: A5 4F      601      LDA    RANDOM+1
002287: E6 4F      602      INC    RANDOM+1
002289: 45 4F      603      EOR    RANDOM+1 ;Leaves 1 if bit changed
00228B: 29 40      604      AND    #%01000000 ;Did bit six change?
00228D: F0 ED =227C 605      BEQ    GETCH4
00228F: D0 E5 =2276 606      BNE    GETCH3  ;Always taken
                                607      ;
002291: F0 9B =222E 608  CTRLB0 BEQ    CTRLR  ;Bounce-back point
                                609      ;
002293: 68          610  GOTKEY PLA
                                ;Remove mask from stack
002294: 68          611      PLA
                                ;Retrieve original char
002295: 91 28      612      STA    (TBASE),Y ;Put it back
002297: AD 00 C0    613      LDA    KEYBD  ;Get key code
00229A: 85 3C      614      STA    KEYCOD  ;Save it for later
                                615      ;
00229C: A2 FF      616      LDX    #$FF
00229E: 2C 61 C0    617      BIT    READOA  ;Check open-apple key
0022A1: 10 04 =22A7 618      BPL    CHKSA
0022A3: 86 43      619      STX    OAFLAG  ;Set open-apple flag
0022A5: A9 8D      620      LDA    #$8D    ;Fake a <RETURN>
                                621      ;
0022A7: 2C 62 C0    622  CHKSA BIT    READSA  ;Check solid-apple key
0022AA: 10 04 =22B0 623      BPL    CHKGET
0022AC: 86 44      624      STX    SAFLAG  ;Set solid-apple flag
0022AE: A9 8D      625      LDA    #$8D    ;Fake a <RETURN>
                                626      ;
0022B0: A6 45      627  CHKGET LDX    SWITCH
0022B2: E0 FE      628      CPX    $FE    ;Is this an &GET?
0022B4: D0 0D =22C3 629      BNE    CHKCAS
0022B6: A6 3C      630      LDX    KEYCOD  ;Get keycode
0022B8: 8E 00 02    631      STX    EDBUF  ;Put it in buffer

```

```

0022BB: E6 42          632          INC      STRLEN      ;Set string_length = 1
0022BD: A9 8D          633          LDA      #$8D        ;Fake a <RETURN>
0022BF: D0 29 =22EA     634          BNE      CONTN1     ;Always taken
                                635          ;
0022C1: 90 8C =224F     636 GET0      BCC      GETCHR      ;Bounce-back point
                                637          ;
0022C3: C9 83          638 CHKCAS    CMP      #$83        ;Check for CTRL-C
0022C5: D0 1F =22E6     639          BNE      CONTIN     ;
0022C7: A6 47          640          LDX      TEMPY      ;Process CTRL-C
0022C9: E4 42          641          CPX      STRLEN     ;Must be char in string
0022CB: B0 17 =22E4     642          BCS      NOALPH     ;Else skip it
0022CD: B1 28          643          LDA      (TBASE),Y ;Get the character
0022CF: 09 20          644          ORA      #$20        ;Force lower case
0022D1: C9 FB          645          CMP      $FEB        ;Hi-ASCII "("
0022D3: B0 0F =22E4     646          BCS      NOALPH     ;Not an alpha
0022D5: C9 E1          647          CMP      $E1        ;Hi-ASCII "a"
0022D7: 90 0B =22E4     648          BCC      NOALPH     ;Not an alpha
0022D9: B1 28          649          LDA      (TBASE),Y ;Retrieve character
0022DB: 49 20          650          EOR      #$20        ;Toggle its case
0022DD: 91 28          651          STA      (TBASE),Y ;Put it back
0022DF: A4 47          652          LDY      TEMPY      ;
0022E1: 99 00 02       653          STA      EDBUF,Y    ;Make change in string
0022E4: A9 95          654 NOALPH    LDA      #$95        ;Hi-ASCII R-ARROW
                                655          ;
0022E6: A2 80          656 CONTIN    LDX      #$80        ;
0022E8: 86 45          657          STX      SWITCH     ;Default SWITCH to $80
0022EA: 8D 54 C0       658 CONTN1    STA      PAGE1     ;Back to pgl if needed
                                659          ;
                                660 *We default to text page 1 because it is
                                661 *assumed that text page 2 was not in use
                                662 *at the time this program was called'. If
                                663 *you wish to work with text page 2 you
                                664 *will have to modify the program.
                                665          ;
0022ED: 8D 10 C0       666          STA      STROBE     ;Clear keyboard strobe
                                667          ;
0022F0: C9 82          668          CMP      #$82        ;Check for CTRL-B
0022F2: D0 02 =22F6     669          BNE      CHKDEL     ;
0022F4: F0 9B =2291     670          BEQ      CTRLB0     ;Always taken
                                671          ;
0022F6: C9 FF          672 CHKDEL    CMP      $FF        ;Check for <DELETE>
0022F8: D0 1C =2316     673          BNE      L13        ;
                                674          ;
                                675 ***PROCESS <DELETE>***
0022FA: AA            676          TAX              ;Leave signature ($FF)
                                677          ;
                                678 ***MOVE CURSOR LEFT***
                                679 *A "signature" in the X register indicates which
                                680 *key processor transferred to CURLFT:
                                681 *X=$88 indicates <L-ARROW>
                                682 *X=$FF indicates <DELETE>
0022FB: A4 47          683 CURLFT    LDY      TEMPY      ;Get string index
0022FD: F0 2B =232A     684          BEQ      REJECT     ;If at L end, no go.
0022FF: C6 47          685          DEC      TEMPY      ;Decrement string index
002301: A4 24          686          LDY      CH        ;Get CH
002303: D0 06 =230B     687          BNE      CURL      ;
002305: C6 25          688          DEC      CV        ;Step up one line
002307: C6 3F          689          DEC      OLDCV     ;Update old CV
002309: A4 21          690          LDY      WNDWID     ;Step to R edge
00230B: 88            691 CURL      DEY              ;Move left 1 character
00230C: 84 24          692          STY      CH        ;Update CH
00230E: 84 40          693          STY      OLDCH     ;Update old CH
002310: 8A            694          TXA              ;Check signature

```

```

002311: 4A          695      LSR          ;Examine LSB
002312: B0 58 =236C  696      BCS   CTRLD  ;Process ctrl-D
002314: 90 AB =22C1    697  GET1    BCC   GET0    ;Always taken
                                698      ;
002316: C9 88          699  L13     CMP   #$88    ;Check <L-ARROW>
002318: D0 03 =231D    700      BNE   CHAR    ;
                                701      ;
                                702  ***PROCESS <L-ARROW>***
00231A: AA          703      TAX          ;Leave signature ($88)
00231B: D0 DE =22FB  704      BNE   CURLFT  ;Always taken
                                705      ;
00231D: C9 A0        706  CHAR    CMP   #$A0    ;Character to insert?
00231F: 90 40 =2361  707      BCC   CONTRL  ;Control character
                                708      ;
                                709  ***PROCESS INSERT***
002321: A4 42          710      LDY   STRLEN
002323: F0 16 =233B  711      BEQ   L16
002325: AA          712      TAX          ;Save char for later
002326: C4 41          713      CPY   FLDLEN  ;STRLEN < FLDLEN?
002328: 90 0C =2336  714      BCC   L15    ;Yes. Continue.
                                715      ;
00232A: 20 DD FB     716  REJECT  JSR   BEEP    ;Beep speaker
00232D: F0 2F =235E  717      BEQ   GETCLC  ;Always taken
                                718      ;
00232F: 88          719  L14     DEY
002330: B9 00 02      720      LDA   EDBUF,Y  ;Open up a hole for
002333: 99 01 02      721      STA   EDBUF+1,Y ;the insertion.
002336: C4 47          722  L15     CPY   TEMPY
002338: D0 F5 =232F  723      BNE   L14
00233A: 8A          724      TXA          ;Retrieve the char
00233B: 99 00 02      725  L16     STA   EDBUF,Y  ;Insert it
00233E: E6 42          726      INC   STRLEN
002340: A2 00          727      LDX   #0      ;Leave signature ($00)
                                728      ;
                                729  ***MOVE CURSOR RIGHT***
002342: A4 47          730      *A "signature" in the X register indicates which
                                731      *key processor transferred to CURRT:
                                732      *X=$00 indicates INSERT
                                733      *X=$95 indicates <R-ARROW>
002344: C4 42          734  CURRT  LDY   TEMPY  ;Get string index
002346: B0 E2 =232A  735      CPY   STRLEN  ;Is TEMPY < STRLEN?
002348: E6 47          736      BCS   REJECT  ;No. Bad news.
00234A: A4 24          737      INC   TEMPY
00234C: C8          738      LDY   CH      ;Get CH
00234D: C4 21          739      INY
00234F: 90 06 =2357  740      CPY   WNDWID  ;CH < WIDTH?
002351: A0 00          741      BCC   L17    ;Yes. Go store it.
002353: E6 25          742      LDY   #0      ;No. Move to next line.
002355: E6 3F          743      INC   CV
002357: 84 24          744      INC   OLDCV
002359: 84 40          745  L17     STY   CH      ;Replace CH
00235B: 8A          746      STY   OLDCH  ;Update old CH
00235C: F0 4F =23AD  747      TXA          ;Retrieve signature
00235E: 18          748      BEQ   REPRNT  ;If called by INSERT
00235F: 90 B3 =2314  749  GETCLC  CLC      ;Force branch
                                750  GET2    BCC   GET1    ;Always taken
                                751      ;
002361: C9 95          752  CONTRL  CMP   #$95  ;Check R-arrow
002363: D0 03 =2368  753      BNE   L18
                                754      ;
                                755  ***PROCESS <R-ARROW>***
002365: AA          756      TAX          ;Leave signature ($95)
002366: D0 DA =2342  757      BNE   CURRT  ;Always taken

```



```

758
002368: C9 84      759 L18    CMP    #$84      ;Check <CTRL-D>
00236A: D0 5B =23C7 760      BNE    L19
761
762 ***PROCESS <CTRL-D>***
00236C: A4 47      763 CTRLD  LDY    TEMPY      ;Get string index
00236E: C4 42      764      CPY    STRLEN      ;Is TEMPY < STRLEN?
002370: B0 B8 =232A 765 REJ1   BCS    REJECT      ;No. Bad news.
002372: B9 01 02     766 CTRLD1 LDA    EDBUF+1,Y  ;Get character to right
002375: 99 00 02     767      STA    EDBUF,Y      ;Store it here
002378: C8          768      INY                ;Step to next character
002379: C4 42      769      CPY    STRLEN      ;Reached end of string?
00237B: D0 F5 =2372 770      BNE    CTRLD1
00237D: A2 00      771      LDX    #0          ;Leave signature ($00)
772
773 ***ERASE STRING***
774 *A "signature" in the X register indicates which
775 *key processor transferred to ERASE:
776 *X=$00 indicates <DELETE> or <CTRL-D>
777 *X=$40 indicates <CTRL-R>
778 *X=$99 indicates <CTRL-Y> or <CTRL-X>
00237F: A5 44      779 ERASE  LDA    TOPCH      ;Get top CH
002381: 85 24      780      STA    CH          ;Put in CH
002383: 8D 7B 05    781      STA    CH80
002386: A5 43      782      LDA    TOPCV      ;Get top CV
002388: 85 25      783      STA    CV          ;Put in CV
00238A: 20 22 FC    784      JSR    VTAB      ;Update TBASE
00238D: A4 42      785      LDY    STRLEN
00238F: F0 08 =2399 786      BEQ    CHKSIG      ;Nothing to erase!
002391: A9 A0      787 ERASE1 LDA    #$A0      ;Hi-ASCII space
002393: 20 ED FD    788      JSR    COUT      ;Print it
002396: 88          789      DEY
002397: D0 F8 =2391 790      BNE    ERASE1
002399: 8A          791 CHKSIG TXA                ;Check signature
00239A: D0 07 =23A3 792      BNE    ERASE2
00239C: C6 42      793      DEC    STRLEN      ;<DELETE> or <CTRL-D>
00239E: D0 0D =23AD 794      BNE    REPRNT
0023A0: 4C 1F 22    795 JMP3   JMP    TESTSW
796
0023A3: 30 04 =23A9 797 ERASE2 BMI    ERASE3
0023A5: 85 45      798      STA    SWITCH      ;<CTRL-R>: $40 to SWITCH
0023A7: D0 04 =23AD 799      BNE    REPRNT      ;Always taken
800
0023A9: A4 47      801 ERASE3 LDY    TEMPY      ;<CTRL-Y>
0023AB: 84 42      802      STY    STRLEN      ;Chop from cursor to end
803
804 ***REPRINT STRING***
805 *A "signature" in the X register indicates which
806 *key processor transferred to REPRNT (via ERASE):
807 *X=$00 indicates INSERT (via CURRT),
808 *<CTRL-D>, or <DELETE>
809 *X=$40 indicates <CTRL-R>
810 *X=$99 indicates <CTRL-Y> or <CTRL-X>
0023AD: A5 44      811 REPRNT LDA    TOPCH      ;Get top CH
0023AF: 85 24      812      STA    CH          ;Put in CH
0023B1: 8D 7B 05    813      STA    CH80
0023B4: A5 43      814      LDA    TOPCV      ;Get top CV
0023B6: 85 25      815      STA    CV          ;Put in CV
0023B8: 20 22 FC    816      JSR    VTAB
0023BB: E0 40      817      CPX    #$40      ;Check signature
0023BD: D0 03 =23C2 818      BNE    REPRN1
0023BF: 4C 64 21    819 JMP1   JMP    REDO      ;<CTRL-R>
0023C2: A0 00      820 REPRN1 LDY    #0

```

```

0023C4: 4C CE 21      821 JMP2      JMP      PRNWRD      ;INSERT, <CTRL-D>,
                                822                                ;<DELETE>, or <CTRL-Y>
                                823                                ;
0023C7: C9 99          824 L19        CMP      #$99      ;Check <CTRL-Y>
0023C9: D0 03 =23CE    825                BNE      L20
                                826                                ;
                                827 ***PROCESS <CTRL-Y>***
0023CB: AA             828                TAX              ;Leave signature ($99)
0023CC: D0 B1 =237F    829 CTRL11     BNE      ERASE      ;Always taken
                                830                                ;
0023CE: C9 92          831 L20        CMP      #$92      ;Check <CTRL-R>
0023D0: D0 06 =23D8    832                BNE      L21
                                833                                ;
                                834 ***PROCESS <CTRL-R>***
0023D2: A2 40          835                LDZ      #$40      ;Leave signature ($40)
0023D4: D0 A9 =237F    836                BNE      ERASE      ;Always taken
                                837                                ;
0023D6: 90 87 =235F    838 GET3       BCC      GET2      ;Bounce-back point
                                839                                ;
0023D8: C9 8A          840 L21        CMP      #$8A      ;Check <D-ARROW>
0023DA: D0 2A =2406    841                BNE      CHKUPA
                                842                                ;
                                843 ***PROCESS <D-ARROW>***
0023DC: A5 25          844                LDA      CV          ;CV < BOTCV?
0023DE: C5 3D          845                CMP      BOTCV
0023E0: 90 02 =23E4    846                BCC      DOWN1
0023E2: B0 8C =2370    847 REJ2       BCS      REJ1      ;No. Bad news.
0023E4: E6 25          848 DOWN1      INC      CV          ;Step down 1 line
0023E6: 18             849                CLC              ;Prepare to add
0023E7: A5 47          850                LDA      TEMPY      ;WNDWID to
0023E9: 65 21          851                ADC      WNDWID      ;TEMPY
0023EB: 85 47          852                STA      TEMPY
0023ED: A5 25          853                LDA      CV          ;CV < BOTCV?
0023EF: 85 3F          854                STA      OLDCV
0023F1: C5 3D          855                CMP      BOTCV
0023F3: 90 0F =2404    856                BCC      DOWN3
0023F5: A5 3E          857                LDA      BOTCH      ;No. Beyond end?
0023F7: C5 24          858                CMP      CH
0023F9: B0 08 =2403    859                BCS      DOWN2
0023FB: 85 24          860                STA      CH          ;Yes. Go back
0023FD: 85 40          861                STA      OLDCH      ;to bottom.
0023FF: A5 42          862                LDA      STRLEN
002401: 85 47          863                STA      TEMPY
002403: 18             864 DOWN2       CLC
002404: 90 D0 =23D6    865 DOWN3      BCC      GET3      ;Always taken
                                866                                ;
002406: C9 8B          867 CHKUPA     CMP      #$8B      ;Check <U-ARROW>
002408: D0 2D =2437    868                BNE      CHKCTX
                                869                                ;
                                870 ***PROCESS <U-ARROW>***
00240A: A5 43          871                LDA      TOPCV      ;TOPCV < CV?
00240C: C5 25          872                CMP      CV
00240E: 90 02 =2412    873                BCC      UPARR1
002410: B0 D0 =23E2    874 REJ3       BCS      REJ2      ;No. Bad news.
002412: C6 25          875 UPARR1     DEC      CV          ;Step up 1 line
002414: A5 25          876                LDA      CV
002416: 85 3F          877                STA      OLDCV
002418: 38             878                SEC
002419: A5 47          879                LDA      TEMPY
00241B: E5 21          880                SBC      WNDWID
00241D: 85 47          881                STA      TEMPY
00241F: A5 43          882                LDA      TOPCV
002421: C5 25          883                CMP      CV          ;TOPCV < CV?

```

```

002423: 90 DF =2404 884      BCC  DOWN3
002425: A5 24      885      LDA  CH      ;No. Left of top?
002427: C5 44      886      CMP  TOPCH
002429: B0 D8 =2403 887      BCS  DOWN2
00242B: A5 44      888      LDA  TOPCH      ;Yes. Go to top.
00242D: 85 24      889      STA  CH
00242F: 85 40      890      STA  OLDCH
002431: A9 00      891      LDA  #0
002433: 85 47      892      STA  TEMPY
002435: F0 CC =2403 893      BEQ  DOWN2      ;Always taken
                                ;
002437: C9 98      895      CHKCTX CMP  #$98      ;Check <CTRL-X>
002439: D0 10 =244B 896      BNE  CHKESC
                                ;
                                897
                                898 ***PROCESS <CTRL-X>***
00243B: AA      899      TAX
00243C: A9 00      900      LDA  #0      ;Go to the top
00243E: 85 47      901      STA  TEMPY
002440: A5 44      902      LDA  TOPCH
002442: 85 40      903      STA  OLDCH
002444: A5 43      904      LDA  TOPCV
002446: 85 3F      905      STA  OLDCV
002448: E8      906      INX      ;$99 to X register
002449: D0 81 =23CC 907      BNE  CTRLY1      ;Always taken
                                ;
00244B: C9 9B      909      CHKESC CMP  #$9B      ;Check <ESC>
00244D: D0 04 =2453 910      BNE  CHKCTN
                                ;
                                911
                                912 ***PROCESS <ESC>***
00244F: E6 46      913      INC  ESCFLG      ;Escape flag to "1"
002451: D0 48 =249B 914      BNE  ESCENT      ;Always taken
                                ;
002453: C9 8E      916      CHKCTN CMP  #$8E      ;Check <CTRL-N>
002455: D0 13 =246A 917      BNE  CHKRTN
                                ;
                                918
                                919 ***PROCESS <CTRL-N>***
002457: A4 42      920      LDY  STRLEN      ;Go to bottom
002459: 84 47      921      STY  TEMPY      ;of string.
00245B: A5 3D      922      LDA  BOTCV
00245D: 85 25      923      STA  CV
00245F: 85 3F      924      STA  OLDCV
002461: A5 3E      925      LDA  BOTCH
002463: 85 24      926      STA  CH
002465: 85 40      927      STA  OLDCH
002467: 18      928      CLC
002468: 90 9A =2404 929      BCC  DOWN3      ;Always taken
                                ;
00246A: C9 8D      931      CHKRTN CMP  #$8D      ;Check <RETURN>
00246C: F0 03 =2471 932      BEQ  RETURN
00246E: 38      933      SEC
00246F: B0 9F =2410 934      BCS  REJ3      ;Always taken
                                ;
                                935
                                936 ***PROCESS <RETURN>***
002471: A5 45      937      RETURN LDA  SWITCH      ;Check &GET
002473: C9 FE      938      CMP  #$FE
002475: F0 08 =247F 939      BEQ  FORMST      ;If &GET, form string
002477: 24 43      940      BIT  OAFLAG      ;Else check for
002479: 30 20 =249B 941      BMI  ESCENT      ;open-apple or
00247B: 24 44      942      BIT  SAFLAG      ;solid-apple
00247D: 30 1C =249B 943      BMI  ESCENT      ;abort of &INPUT.
                                ;
00247F: A6 42      945      FORMST LDX  STRLEN      ;Get string length
002481: F0 0A =248D 946      BEQ  RTN2      ;Length = zero?

```

```

002483: A9 A0          947      LDA    # $A0      ;Delete trailing spaces
002485: DD FF 01          948      RTN1    CMP    EDBUF-1,X  ;Is character a space?
002488: D0 03 =248D       949      BNE    RTN2    ;No. Go create string.
00248A: CA                950      DEX                ;Yes. Strip it off.
00248B: D0 F8 =2485       951      BNE    RTN1    ;Go back if more chars
                                952      ;
00248D: 20 39 D5         953      RTN2    JSR    GDBUFS    ;Form string in EDBUF
                                954      ;
                                955      *GDBUFS puts a null ($00) at the
                                956      *end of the string in EDBUF and
                                957      *masks off the MSB of all bytes.
                                958      *GDBUFS expects string_length in X
                                959      *GDBUFS returns with (A)=0, (Y)=1
                                960      ;
002490: C8                961      INY                ;(Y,A) set to $200
002491: 85 0D             962      STA    CHARAC    ;No other terminator
002493: 85 0E             963      STA    ENDCHR    ;except a null byte.
002495: 20 ED E3         964      JSR    STRLT2    ;Form temporary string
                                965      ;
                                966      *STRLT2 expects Y,A to point to
                                967      *a literal low-ASCII string. A
                                968      *temporary string is formed in
                                969      *memory space that is requested
                                970      *below FRETOP. In addition to
                                971      *the null ($00) terminator, the
                                972      *values in CHARAC and ENDCHR
                                973      *are used as string terminators.
                                974      ;
002498: 20 7B DA         975      JSR    PERMST    ;Make it permanent
                                976      ;
00249B: A9 00          977      ESCENT    LDA    #0      ;<ESC> enters here
00249D: 85 10          978      STA    DIMFLG    ;Initialize flags
00249F: 85 11          979      STA    VALTYP
0024A1: 85 12          980      STA    INTFLG
0024A3: A9 4F          981      LDA    #$4F      ;Lo-ASCII 'O'
0024A5: 85 81          982      STA    VARNAM
0024A7: A9 41          983      LDA    #$41      ;Lo-ASCII 'A'
0024A9: 85 82          984      STA    VARNAM+1
0024AB: 20 4F E0       985      JSR    VARLOC    ;Locate the variable OA
0024AE: 85 85          986      STA    FORPNT    ;Aim FORPNT at the
0024B0: 84 86          987      STY    FORPNT+1  ;variable value.
0024B2: A0 00          988      LDY    #0      ;Default OA to zero
0024B4: 24 43          989      BIT    OAFLAG
0024B6: 10 02 =24BA    990      BPL    FLTOA
0024B8: A4 3C          991      LDY    KEYCOD    ;If flag, use KEYCODE
0024BA: 20 01 E3       992      FLTOA    JSR    SNGFLT    ;Float new OA value
0024BD: 20 27 EB       993      JSR    STORE    ;Store it in OA
                                994      ;
0024C0: A9 53          995      LDA    #$53      ;Lo-ASCII 'S'
0024C2: 85 81          996      STA    VARNAM
                                997      ;
                                998      ;(VARNAM+1 still holds 'A')
                                999      ;
0024C4: 20 4F E0       1000     JSR    VARLOC    ;Locate the variable SA
0024C7: 85 85          1001     STA    FORPNT    ;Aim FORPNT at the
0024C9: 84 86          1002     STY    FORPNT+1  ;variable value.
0024CB: A0 00          1003     LDY    #0      ;Default SA to zero
0024CD: 24 44          1004     BIT    SAFLAG
0024CF: 10 02 =24D3    1005     BPL    FLTSA
0024D1: A4 3C          1006     LDY    KEYCOD    ;If flag, use KEYCODE
0024D3: 20 01 E3       1007     FLTSA    JSR    SNGFLT    ;Float new SA value
0024D6: 20 27 EB       1008     JSR    STORE    ;Store it in SA
                                1009     ;

```

0024D9: A9 45	1010	LDA	#\$45	;Lo-ASCII 'E'
0024DB: 85 81	1011	STA	VARNAM	
0024DD: A9 53	1012	LDA	#\$53	;Lo-ASCII 'S'
0024DF: 85 82	1013	STA	VARNAM+1	
0024E1: 20 4F E0	1014	JSR	VARLOC	;Locate the variable ES
0024E4: 85 85	1015	STA	FORPNT	;Aim FORPNT at the
0024E6: 84 86	1016	STY	FORPNT+1	;variable value.
0024E8: A4 46	1017	LDY	ESCFLG	; "1" if <ESC>, else "0"
0024EA: 20 01 E3	1018	JSR	SNGFLT	;Float new ES value
0024ED: 20 27 EB	1019	JSR	STORE	;Store it in ES
	1020			;
0024F0: A5 3D	1021	LDA	BOTCV	;Move cursor to
0024F2: 85 25	1022	STA	CV	;bottom of display
0024F4: 20 22 FC	1023	JSR	VTAB	; (one character position
0024F7: A5 3E	1024	LDA	BOTCH	;beyond last char in
0024F9: 85 24	1025	STA	CH	;string, including
0024FB: 8D 7B 05	1026	STA	CH80	;trailing spaces),
0024FE: 60	1027	RTS		;and exit.

The Gentleman's GS: A Polite Introduction to the 16-bit II

Part II

by Ross W. Lambert



Last month we eased into a few definitions and a cursory examination of the tool startup order. I finished by suggesting that we'll "revisit" that demonic (for me) piece of code I called Generic Start.

Let me preface that visitation by saying that the GS can be a time bomb. It really pays to learn how to do things right the first time because erroneous code might not produce problems right away (believe me, I know from *experience*, positively embarrassing experience at that, as y'all know). Your program might actually crash in a section of code far removed from the point of the error. Some programs might not crash at all - right away. They save their explosions for an opportune time (opportune being defined as that moment in which a crash will cause the most distressing mischief).

This has always been the case with assembly code (aw heck, it's true in *any* programming environment), but it is particularly pervasive in *my* assembly language GS programs. The reason? I mentioned it briefly last month: the method Apple chose for passing parameters to and from the toolbox is to place them on top of the stack. This is not a bad thing, really, but if you don't watch your pushes and pulls (PHAS and PLAS or PushWords and PullWords, etc.), you can get them out of balance. Since many of the tool calls require multiple parameters of various sizes, it is easier to screw them up than you might think. If you return from a subroutine with an extraneous parameter squatting astride the stack, for example, your program will try to return to the wrong address. It is more than likely that you will be teleported into oblivion.

That said, we can attack the startup procedure again. Let's take it one step at a time.

A quick stroll down memory lane

First, a fact: the GS memory is organized into 64K banks. Like the main mem and aux mem switching from days of old, you can have a program running in one bank that reads and writes data in another. For the purposes of startup, however, your program will usually want to read data from and write data to the same bank in which it lives.

Unlike the good ol' 8-bit days (?) when you read a softswitch or two, the 65816 CPU has a few new appendages which determine where the processor looks for instructions and data. These new limbs are called the program bank register and the data bank register.

Getting the program bank and the data bank to be one and the same can be accomplished by grabbing the value of the program bank register and pushing it onto the stack. Then, in a not so subtle manipulation, yank the bugger back off the stack and stuff it into the data bank register.

This effectively makes the data bank equal to the program bank. It is a maneuver you'll see often in GS code, and looks like this:

```
Start   phk      ;push program bank register.  
        plb      ;pull back into data bank register.
```

You might be wondering why you cannot set the data bank directly, akin to switching between main and auxiliary memory on a IIC or 128K IIC. The reason is that GS programs don't really need to know where they live, at least not very often. The Memory Manager takes care of that. Programs are therefore relocatable and have to set things like data banks indirectly (like the method used above).

An aside - before I started working with the GS (last fall - yes, I *am* new at this, but I think I'm living, breathing proof that a rank beginner can really have good time with the machine), I thought that writing relocatable code for the GS meant jumping through all of the same hoops that it did for the 8 bit Apples. I thought I could never reference labels within my own program, for example. But lo and behold, Apple created a beast called the OMF (Object Module Format). This object code format includes a relocating dictionary which helps the GS (the system loader, actually) relocate your code on its own! Instead of writing your own relocater module or forcing your code to be absolutely and purely relocatable ala' the 8-bit world, the system worries about it for you.

You can write fixed position code for the GS if you really want to since the design team built in all kinds of flexibility into the memory manager. But since relocation worries are pretty much behind us, it is almost pointless.

Notice I said "almost". There are times and instances, I can imagine, wherein carefully crafted, fixed position code could blow the socks off standard OMF performance. But the instances are few and the disadvantages outweigh the advantages for all of the applications I'm inclined to write. (Incidentally and FYI - although I don't recommend the idea, Microl Systems of Canada has created their own proprietary "fastload" object code format which greatly speeds up the rate at which a program is plopped into memory. There is, as they say, more than one way to skin a cat.)

Back to our subject. The next step in the startup process is to start the Tool Locator. This is *always* the first tool started because it is the bus that all the others ride. We're dead in the water without it, if you'll excuse mixed metaphors.

The code looks like this:

```
_TLStartUp      ;start tool locator
```

Roger is different...

If you own Roger Wagner's *Apple IIGS Assembly Language Programming for Beginners*, you'll notice that the Tool Locator startup looks like this instead:

```
LDX #0201      ;Tool Locator StartUp call number  
JSL $E10000    ;tool call entry point
```

This example is taken from p. 321, if you care to look it up. The reason for the apparent discrepancy is that my `_TLStartUp` is a macro name. The macro creates Roger's expanded code immediately above this paragraph. Roger discussed creating your own tool macros in the book, the reason being that the text must've been written before the Merlin disk included all of the Tool.Macros macro libraries. I'm certainly glad they are there now!

Needless to say, it is much easier to work with the macro names than to do tool calls "by hand". Remembering the tool call numbers is next to impossible. But now you know that the macros at least include code to load the X register with the tool number and do a long jump (i.e. between 64K banks) to the subroutine that handles toolbox calls.

A tilde for Hilda...

There's yet another class of macros on the recent Merlin disks, these by Dave Klimas (for you APW folks, there is a set of identical macros available from PunkWare, P.O. Box 874043, Wasilla, AK 99687-4073. Send \$15 and ask for "PW Macros"). Called tilde macros because they're prescripted with the tilde character (~), they combine all of the "pushes" for parameter passing into one step. We'll look at these in more detail later in this series. Some programmers swear by them, but I think beginners like me need to grow into them. I find myself forgetting whether I'm working with single bytes, words (two bytes), or long words (four bytes). The tilde macros can make debugging a little more complicated for me because I cannot readily see the size of the parameter I pushed on the stack. Once you've got a given tool call down pat, though, you may grow weary of typing all of the PHAs, PushWords or PushLongs. That being the case, you're ready for Dave's macros.

The Tool Locator toolset is a permanent resident of your GS - it's in ROM. In this respect it is different than most of the other toolsets. But we'll get to that next month.

Until then, then.



The Sourceror's Apprentice

Copyright (C) 1989 by Ross W. Lambert and Ariel Publishing

All programs in **The Apprentice** are in the public domain and may be freely copied and distributed, but NOT sold. Apple User Groups and other important folks may reprint articles upon request. Just gimme a call at 509/923-2025 or drop me a line at P.O. Box 398, Pateros, WA 98846.

American prices in US Dollars... 1 year \$28, 2 years \$52

Canadian and Mexican subscribers add \$5 per year, all other non-North American subscribers add \$15 per year for first class postage.

Editor and Publisher..... Ross W. Lambert

Technical Editors (and moral support)..... Eric Mueller, Jay Jennings, Robert Moore

Subscription Services.....Tamara Lambert, Cindy Eckels

Stamp Licking..... Rebecca Lambert

WARRANTY and LIMITATION of LIABILITY

I warrant that the information in **The Apprentice** is correct and useful to somebody somewhere. Any subscriber may ask for a full refund of their last subscription payment at any time. My LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, or for ANY damages in excess of the fees paid by a subscriber.

The Sourceror's Apprentice is a product of the United States of America.

Apple, Apple II, Apple IIGS, ProDOS, and BASIC.SYSTEM are registered trademarks of Apple Computers, Inc.

Ariel Publishing
Box 398
Pateros, WA 98846